Simple and Efficient Real Root-finding for a Univariate Polynomial

Victor Y. Pan, Elias Tsigaridas, Zhao Liang

Simple and Efficient Real Root-finding for a Univariate Polynomial

Victor Y. Pan [1,2],[a]*, Elias P. Tsigaridas[3]†, and Liang Zhao[2],[b]

[1] Department of Mathematics and Computer Science
Lehman College of the City University of New York
Bronx, NY 10468 USA
[2] Ph.D. Programs in Mathematics and Computer Science
The Graduate Center of the City University of New York
New York, NY 10036 USA
[a] victor.pan@lehman.cuny.edu
http://comet.lehman.cuny.edu/vpan/
[b] lzhao1@gc.cuny.edu
[3] INRIA, Paris-Rocquencourt Center, *PolSys*
Sorbonne Universités, UPMC Univ. Paris 06, *PolSys*,
UMR 7606, LIP6, F-75005, Paris, France
elias.tsigaridas@inria.fr

Abstract

Univariate polynomial root-finding is a classical subject, still important for modern computing. Frequently one seeks just the real roots of a polynomial with real coefficients. They can be approximated at a low computational cost if the polynomial has no nonreal roots, but for high degree polynomials, nonreal roots are typically much more numerous than the real ones. The challenge is known for long time, and the subject has been intensively studied. Nevertheless, we obtain dramatic acceleration of the known algorithms by applying new combinations of the known algorithms and properly exploiting the geometry of the complex plane. We confirm the efficiency of the proposed real root-finders by both their Boolean complexity estimates and the results of their numerical tests with benchmark polynomials. In particular in our tests the number of iterations required for convergence of our algorithms grew very slowly as we increased the degree of the polynomials from 64 to 1024. Our techniques is very simple, and we point out their further modifications that promise to produce efficient complex polynomial root-finders.

Keywords: Polynomials
Real roots
Root radii

1 Introduction

Assume a univariate polynomial of degree n with real coefficients,

$$p(x) = \sum_{i=0}^{n} p_i x^i = p_n \prod_{j=1}^{n} (x - x_j), \quad p_n \neq 0, \tag{1.1}$$

*VP and LZ have been supported by NSF Grant CCF 1116736 and by PSC CUNY Award 67699-00 45.
†ET has been partially supported by GeoLMI (ANR 2011 BS03 011 06), HPAC (ANR ANR-11-BS02-013) and an FP7 Marie Curie Career Integration Grant

which has r real roots $x_1, \dots, x_r$ and $s = (n-r)/2$ pairs of nonreal complex conjugate roots. In some applications, e.g., to algebraic and geometric optimization, one seeks only the r real roots, which make up just a small fraction of all roots. This is a well studied subject (see [EPT14, Section 10.3.5], [PT13], [SMa], and the bibliography therein), but the most popular numerical packages of subroutines for root-finding such as MPSolve 2.0 [BF00], Eigensolve [F02], and MPSolve 3.0 [BR14] approximate the r real roots about as fast and as slow as all the n complex roots.

It can be surprising, but by combining some well known but well ignored algorithms for the approximation the root radii, that is, the distances of the roots to the origin, with Dandelin's classical root-squaring iteration [H59], and properly exploiting the geometry of the complex plane, we accelerate the solution by a factor of n/r, which means dramatic speed up in the cited important applications. Ee confirm their efficiency with the estimates for their Boolean complexity and the results of our numerical tests, in which the number of iterations required for convergence of our algorithms grew very slowly as we increased the degree of the polynomials from 64 to 1024. Our techniques is very simple, and we point out their further modifications that promise to produce efficient complex polynomial root-finders. We organize our paper as follows. In the next section ,,,,,,,,,,,,,,,,,.....

our algorithms, In Section 3 we estimate their Boolean complexity. In Section 4 we present the results of our numerical tests.

2 Real Polynomial Root-finding by Means of the Root-radii Approximation

Hereafter "flop" stands for "arithmetic operation". $O_B(\cdot)$ and $\tilde{O}_B(\cdot)$ denote the Boolean complexity up to some constant and polylogarithmic factors, respectively.

2.1 Some Maps of the Variables and the Roots

Some basic maps of polynomial roots can be computed at a linear or nearly linear arithmetic cost.

Theorem 2.1. *(*Root Inversion, Shift and Scaling*, cf. [P01].)*

(i) Given a polynomial $p(x)$ of (1.1) and two scalars a and b, one can compute the coefficients of the polynomial $q(x) = p(ax+b)$ by using $O(n \log(n))$ flops. This bound decreases to $2n-1$ multiplications if $b = 0$.

(ii) Reversing a polynomial inverts all its roots involving no flops, that is, $p_{\rm rev}(x) = x^n p(1/x) = \sum_{i=0}^{n} p_i x^{n-i} = p_n \prod_{j=1}^{n}(1 - x x_j)$.

Note that by shifting and scaling the variable, we can move all roots of $p(x)$ into a fixed disc, e.g., $D(0,1) = \{x : \ |x| \le 1\}$.

Theorem 2.2. *(*Dandelin's Root Squaring*, cf. [H59].)*

(i) Let a polynomial $p(x)$ of (1.1) be monic. Then $q(x) = (-1)^n p(\sqrt{x}\,) p(-\sqrt{x}\,) = \prod_{j=1}^{n}(x - x_j^2)$.

(ii) One can evaluate $p(x)$ at the k-th roots of unity for $k > 2n$ and then interpolate to $q(x)$ by using $O(k \log(k))$ flops overall.

Remark 2.1. *Recursive root-squaring is prone to numerical stability problems because the coefficients of the iterated polynomials very quickly span many orders of magnitude. Somewhat surprisingly, the Boolean complexity of the recursive root-squaring process is still reasonable if high output precision is required [P95], [P02], and we confirm and strengthen this observation with our new study. Note also that one can avoid the numerical stability problems and perform all iterations with the standard IEEE double precision by applying a special tangential representation and renormalization of the coefficients and the intermediate results proposed in [MZ01]. In this case the computations involve more general operations than flops, and in terms of the CPU time the computational cost per iteration has the same order as n^2 flops.*

2.2 Counting the Roots in a Disc. Root Radii, Distances to the Roots, and the Proximity Tests

In this subsection we estimate the distances to the roots of $p(x)$ from a complex point and the number of roots in an isolated disc.

Hereafter a disc $D(X,r)$ is said to be γ-*isolated* for a polynomial $p(x)$ and $\gamma > 1$ if it contains all roots of the polynomial lying in the disc $D(X,\gamma r)$. In this case we say that the disc has the *isolation ratio* at least γ.

The number of roots in an isolated disc can be computed by using the following result from [R87, Lemma 7.1] (cf. also [S82, Theorem 14.1]).

Theorem 2.3. *[R87, Lemma 7.1] It is sufficient to perform FFT at $n' = 16\lceil \log_2 n \rceil$ points (using $1.5n' \log(n')$ flops) and $O(n)$ additional flops and comparisons of real numbers with 0 in order to compute the number of roots of a polynomial $p(x)$ of (1.1) in a 9-isolated disc $D(0,r)$.*

Remark 2.2. *The algorithm of [R87] supporting Theorem 2.3 only uses the signs of the real and imaginary parts of the n output values of FFT. For some groups of the values, the pairs of the signs stay invariant and can be represented by a single pair of signs. Can this observation be exploited in order to decrease the computational cost of performing the algorithm?*

Corollary 2.1. *It is sufficient to perform $O(hn\log(n))$ flops and $O(n)$ comparisons of real numbers with 0 in order to compute the number of roots of a polynomial $p(x)$ of (1.1) in an s-isolated disc $D(0,r)$ for $s = 9^{1/2^h}$ and for any positive integer h.*

Proof. Every root-squaring of Theorem 2.2 squares all root-radii and the isolation ratii of all discs $D(0,r)$. Suppose h repeated squaring iterations map a polynomial $p(x)$ into $p_h(x)$, for which the disc $D(0,1)$ is 9-isolated. Then we can compute the number of roots of $p_h(x)$ in this disc by applying Theorem 2.3, which is the same as the number of roots of $p(x)$. □

In view of Remark 2.1, one must apply the slower operations of [MZ01] or high precision computations in order to support even a moderately long sequence of root-squaring iterations, but in some cases it is sufficient to apply Corollary 2.1 for small positive integers h. Note that $9^{1/2^h}$ is equal to 1.3160... for $h = 2$, to 1.1472... for $h = 3$, to 1.0710... for $h = 4$ and to 1.0349... for $h = 5$.

We can use the following result if we agree to perform computations with extended precision.

Theorem 2.4. *(*The Root Radii Approximation.*)*

Assume a polynomial $p(x)$ of (1.1) and two real scalars $c > 0$ and d. Define the n root radii *$r_j = |x_{k_j}|$ for $j = 1,\dots,n$, distinct $k_1,\dots,k_n$, and $r_1 \ge r_2 \ge \cdots \ge r_n$. Then, by using $O(n\log^2(n))$ flops, one can compute n approximations $\tilde r_j$ to the root radii r_j such that $\tilde r_j \le r_j \le (1 + c/n^d)\tilde r_j$, for $j = 1,\dots,n$.*

Proof. (Cf. [S82], [P00, Section 4].) At first fix a sufficiently large integer k and apply k times the root-squaring of Theorem 2.2, which involves $O(kn\log(n))$ flops. Then apply the algorithm of [S82] to approximate all root radii $r_j^{(k)} = r_j^{2^k}$, $j = 1,\dots,n$, of the output polynomial $p_k(x)$ within a factor of $2n$ by using $O(n)$ flops. By taking the 2^k-th roots, approximate the root radii $r_1,\dots,r_n$ within a factor of $(2n)^{1/2^k}$, which is $1 + c/n^d$ for k of order $\log(n)$. □

Alternatively we can approximate the root radii by employing the Gerschgörin theorem to the companion or generalized companion matrices of a polynomial $p(x)$ [C91], by applying the heuristic method of [B96], used in the packages MPSolve 2000 and 2012 [BF00], [BR14], or by recursively applying Theorem 2.3, although neither of these techniques support competitive complexity estimates.

The following two theorems define the largest root radius r_1 of the polynomial $p(x)$.

Theorem 2.5. *(See [VdS70].) Assume a polynomial $p(x)$ of (1.1). Write $r_1 = \max_{j=1}^n |x_j|$, $r_n = \min_{j=1}^n |x_j|$, and $\gamma^+ = \max_{i=1}^n |p_{n-i}/p_n|$. Then $\gamma^+/n \le r_1 \le 2\gamma^+$.*

Theorem 2.6. *(See [P01a].) For $\epsilon = 1/2^b > 0$, one only needs $a(n,\epsilon) = O(n + b\log(b))$ flops to compute an approximation $r_{1,\epsilon}$ to the largest root radius r_1 of $p(x)$ such that $r_{1,\epsilon} \le r_1 \le 5(1+\epsilon)r_{1,\epsilon}$. In particular, $a(n,\epsilon) = O(n)$, for $b = O(n/\log(n))$, and $a(n,\epsilon) = O(n\log(n))$, for $b = O(n)$.*

Both theorems can be immediately extended to the approximation of the smallest root radius r_n because it is the largest root radius of the reverse polynomial $p_{\rm rev}(x) = x^n p(1/x)$ (cf. Theorem 2.1). Moreover, by shifting a complex point c into the origin (cf. Theorem 2.1), we can turn our estimates for the root radii into the estimates for the *distances to the roots* from the point c. Approximation of the smallest distance from a complex point c to a root of $p(x)$ is called the *proximity test* at the point. One can perform such a test by applying Theorems 2.3, 2.5, or 2.6.

Alternatively, for proximity tests *by action* at a point c or at n points, one can apply Newton's iterations

$$y_0 = c,\ y^{(h+1)} = y^{(h)} - p(y^{(h)})/p'(y^{(h)}),\ h = 0,1,\dots \tag{2.1}$$

and estimate the distance to the roots by observing convergence or divergence of the iterations.

Theorem 2.6 and these iterations can be applied even where a polynomial $p(x)$ is defined by a black box subroutine for its evaluation rather than by its coefficients.

2.3 A Real Root-finder Based on the Root-radii Approximation

Algorithm 2.1. Real root-finding by means of root radii approximation.

INPUT: *two integers n and r, $0 < r < n$, and the coefficients of a polynomial $p(x)$ of equation (1.1).*

OUTPUT: *approximations to the real roots $x_1,\dots,x_r$ of the polynomial $p(x)$ or FAILURE with a probability close to 0.*

COMPUTATIONS:

1. *Compute approximations $\tilde r_1,\dots,\tilde r_n$ to the root radii of a polynomial $p(x)$ of (1.1) (see Theorem 2.4). (This defines $2n$ candidates points $\pm\tilde r_1,\dots,\pm\tilde r_n$ for the approximation of the r real roots $x_1,\dots,x_r$.)*

2. *At all of these $2n$ points, apply one of the proximity tests of Section 2.2, to select r approximations to the r real roots of the polynomial $p(x)$.*

3. *Apply Newton's iteration $x^{(h+1)} = x^{(h)} - p(x^{(h)})/p'(x^{(h)})$, $h = 0,1,\dots$, concurrently at these r points, expecting to refine quickly the approximations to the isolated simple real roots.*

One can ensure numerical stability of computations at Stage 1 by applying the techniques of [MZ01] for root squaring iteration. Can we accelerate the computations by applying the algorithm of Theorem 2.3 and observations of Remark 2.2?

Remark 2.3. *(*Refinement by means of Newton's iteration.*) For every h, $h = k, k-1,\dots,0$, we can apply concurrently Newton's iteration $x^{(h)}_{j,i+1} = x^{(h)} - p(x^{(h)}_{j,i})/p'(x^{(h)}_{j,i})$, for $i = 0,1,\dots,l$ (cf. (2.1)), at the r approximations $x^{(h)}_{j,0} = x^{(h)}_j$, $j = 1,\dots,r$, to the r real roots of the polynomial $p_h(x)$. We can perform an iteration loop by using $O(n\log^2(r))$ flops, that is, $O(nl\log^2(r))$ flops in l loops (cf. [P01, Section 3.1]), and include these flops into the overall arithmetic cost of order $kn\log(n)$ for performing the algorithm. We can perform the proximity tests of Stage 4 of the algorithm by applying Newton's iteration at all $2r$ candidate approximation points. Having selected r of them, we can continue applying the iteration at these points, to refine the approximations.*

Remark 2.4. *(*Handling the Nearly Real Roots.*) The integer parameter k and the overall arithmetic cost of performing the algorithm are large if the value $2^{-d} = \min_{j=r+1}^{n} |\Im x_j|$ is small. We can counter this deficiency by splitting out the factor $v_{k,+}(x)$ of the polynomial $t_k(x)$ having degree $r_+ > r$ and thus having r real and $r_+ - r$ nearly real roots. (Clearly we can select readily the r real roots among the r_+ real and $r_+ - r$ nearly real roots, and we assume that the other nonreal roots of the polynomial $t_k(x)$ lie much farther from the real axis.) Our convergence analysis and the recipes for splitting out the factor $v_k(x)$ (including the previous remark) can be readily extended. If the integer r_+ is small,*

we can compute all the r_+ roots of the polynomial $v_{k,+}(x)$ at a low cost, but even if the integer r_+ is large, but all of r_+ roots of the polynomial $v_{k,+}(x)$ lie on or close enough to the real axis, we can approximate these roots at a low cost by applying the modified Laguerre algorithm of [DJLZ97].

3 On the Boolean complexity of Algorithm 2.1

We need the following lemma from [PT13, PT14c] on polynomial multiplication.

Lemma 3.1. *Let $A, B \in \mathbb{C}[x]$ of degree at most d, such that $\|A\|_\infty \le 2^{\tau_1}$ and $\|B\|_\infty \le 2^{\tau_2}$. Let C denote the product AB and let $K = 2^k \ge 2d+1$ for a positive integer k. Write $\lambda = \ell + 2\tau_1 + 2\tau_2 + 5.1 \lg K + 4$. Assume that we know the coefficients of A and B up to the precision λ, that is, that the input includes two polynomials $\widetilde{A}$ and $\widetilde{B}$ such that $\|A - \widetilde{A}\|_\infty \le 2^{-\lambda}$ and $\|B - \widetilde{B}\|_\infty \le 2^{-\lambda}$. Then we can compute in $O_B(d \lg d\, \mu(\ell + \tau_1 + \tau_2 + \lg d))$ a polynomial $\widetilde{C}$ such that $\|C - \widetilde{C}\|_\infty \le 2^{-\ell}$. Moreover, $\|C\|_\infty \le 2^{\tau_1+\tau_2+2\lg K}$ for all i.*

Remark 3.1. *In the sequel, for simplicity we occasionally replace the value $\lambda = \ell + 2\tau_1 + 2\tau_2 + 5.1 \lg(2d+1) + 4$ by its simple upper bound $\ell + 2\tau_1 + 2\tau_2 + 6 \lg d + 15$.*

We need the following result from [S82, Theorem 19.1] .

Theorem 3.1. *Let f be polynomial of degree n with all its roots in the unit disc. Let $\widetilde{f}$ be a λ-approximation, that is, $\|f - \widetilde{f}\|_\infty \le 2^{-\lambda}$. Then the roots of f, $\alpha_1, \ldots, \alpha_n$, and the roots of $\widetilde{f}$ can be numbered such that, for $j \in [n]$,*

$$|\alpha_j - \widetilde{\alpha}_j| \le 2^{-\lambda/n + 2\lg(n)/n + 2}.$$

If the roots are bounded by r, then a term $\lg(r)$ should be added in the exponent.

3.1 The Complexity of Root Radii Approximations

Given a polynomial $f(x)$, Dandelin's root-squaring operator is the following map,

$$\mathcal{D} : f(x) \mapsto (-1)^n f(-\sqrt{x}) f(\sqrt{x})$$

where $y = x^2$ (cf. Theorem 2.2). By using the representation with the square roots we obtain the output polynomial of the same degree as f.

We need the following lemma, which bounds the propagation errors and the height of the polynomials computed in a sequence of Dandelin's iterations.

Lemma 3.2. *Let $f \in \mathbb{C}[x]$ be a polynomial of degree d such that $\|f\|_\infty \le 2^\tau$ Let the polynomial f_k be output in k Dandelin's root-squaring iterations applied to f, and let $N = 2^k$.*

Let $\widetilde{f}$ be a λ-approximation of f, that is, $\|f - \widetilde{f}\|_\infty \le 2^{-\lambda}$, where $\lambda = \ell + 4(2^k - 1)\tau + (8 \cdot 2^k - 2k - 8)d - 16 \cdot 2^k - k - 16 = -\lambda + O(N\tau + N\,d)$. We can obtain an ℓ-approximation to f_k, $\widetilde{f}_k$, such that $\|f_k - \widetilde{f}_k\|_\infty \le 2^{-\ell}$ in $O_B(k\,d\,\lg d\,\mu(\ell)) = \widetilde{O}_B(k\,d\,(N\tau + Nd))$.

Moreover, $\lg\|f_k\|_\infty \le 2^k \tau + (2^{k+1} - 2)\lg(d) + 4 \cdot 2^k - 4 = O(N\tau + N \lg d)$.

Proof. We prove the upper bound on the height by induction. For $k = 1$, we perform the multiplication $f_1(x) = f(-x)f(x)$. Lemma 3.1 implies that $\|f_1\|_\infty \le 2^{h(1)}$ where $h(1) = 2\tau + 2\lg d + 4$, which agrees with our bound.

Assume that the claimed bound holds up to $k-1$. At step k we perform the multiplication $f_{k-1}(-x)\, f_{k-1}(x)$. By the induction hypothesis, $\|f_{k-1}\|_\infty \le 2^{h(k-1)}$ where $h(k-1) = 2^{k-1}\tau + (2^k - 2)\lg(d) + 4 \cdot 2^{k-1} - 4$. By applying Lemma 3.1 we obtain the desired bound.

Next we prove the bound on the approximation errors. Let $E(k)$ be the approximation output by the k-th iteration. For example, $\|f - \widetilde{f}\|_\infty = \|f_0 - \widetilde{f}_0\|_\infty \le 2^{E(0)}$ and $E(0) = -\lambda$. Notice that $E(k) = E(k-1) + 4\,h(k-1) + 6 \lg d + 15$. The solution of this recurrence relation provides us with the error bound. □

The approximation of all the root radii of a polynomial corresponds to solving the following task, for all $s \in [n]$, where n is the degree of the polynomial (see [S82, MP13]).

Task S. Given a positive Δ and an integer $s \in [n]$ find a positive r such that $r/(1+\Delta) < r_s < (1+\Delta)r$.

Assume that we have solved Task S for a λ-approximation $\widetilde{f}$ to a polynomial f such that $\|f - \widetilde{f}\|_\infty \le 2^{-\lambda}$, and now would like to extend this solution to the solution of Task S for the polynomial f. The following lemma links the approximation errors of this extension with its Boolean cost and the value of λ.

Lemma 3.3. *Let $f \in \mathbb{C}[x]$ have degree n such that $\|f\|_\infty \le 2^\tau$ and have all its roots lying inside a disc in the complex plane centered at the origin with radius 2^τ. Assume that we are given a λ-approximation of f, $\widetilde{f}$, such that $\|f - \widetilde{f}\|_\infty \le 2^{-\lambda}$, where $\lambda = \ell + O(n\tau + n^2)$, $\ell > 0$.*

Then we can solve Task S for f and all $s \in [n]$ by using $\widetilde{f}$, with $1/\Delta \le d^{O(1)}$ in $\widetilde{O}_B(n^3 + n^2\tau + n\ell)$.

Proof. We wish to solve Task S for $\widetilde{f}$. At first assume that $1 + \Delta \ge 2n$ and compute an r such that

$$\frac{r}{1+\Delta} \le \widetilde{r}_s \le (1+\Delta)r. \tag{3.1}$$

(We can apply the algorithm supporting [S82, Theorem 14.2], cf. [MP13, Algorithm 15.4.1].) Let us deduce from these inequalities that

$$\frac{r}{2(1+\Delta)} \le r_s \le 2(1+\Delta)r \tag{3.2}$$

provided that λ is small enough.

The application of the perturbation theorem (Theorem 3.1) to f and $\widetilde{f}$ results in the bounds

$$|r_s - \widetilde{r}_s| \le |\alpha_s - \widetilde{\alpha}_s| \le 2^{-\lambda/n + 2\lg(n)/n + 2 + \tau}$$

where α_j are the roots of f and $\widetilde{\alpha}_j$ are the roots of $\widetilde{f}$. Now recall that $\rho = \lambda/n - \lg(n)/n - 2 - \tau$ and deduce that

$$\widetilde{r}_s - 2^{-\rho} \le r_s \le \widetilde{r}_s + 2^{-\rho}. \tag{3.3}$$

Next assume that the left or the right inequality of (3.2) does not hold, show a contradiction for a sufficiently small value of λ, and estimate for which larger values λ these inequalities still hold.

At first assume that the left inequality of (3.2) does not hold, that is, $\frac{1}{2(1+\Delta)} > r_s$. By combining the left inequalities of (3.3) and (3.1) obtain the bounds

$$\frac{r}{1+\Delta} - 2^{-\rho} \le \widetilde{r}_s - 2^{-\rho} \le r_s \le \frac{r}{2(1+\Delta)},$$

which imply that

$$\frac{r}{1+\Delta} - 2^{-\rho} \le \frac{r}{2(1+\Delta)} \Rightarrow \frac{r}{2(1+\Delta)} \le 2^{-\rho} \Rightarrow \rho \le \lg(1+\Delta) + 1 - \lg r.$$

Taking into account that $\rho = \lambda/n - \lg(n)/n - 2 - \tau$, obtain $\lambda \le n\lg(1+\Delta) + 3n - n\lg r + \lg n$.

So we must assume that

$$\lambda > n\lg(1+\Delta) + 3n - n\lg r + \lg n + n\tau. \tag{3.4}$$

Now suppose that the right inequality of (3.2) does not hold, that is, $r_s > 2(1+\Delta)r$. Then combine (3.3) and (3.1) and deduce that

$$2(1+\Delta)r \le r_s \le \widetilde{r}_s + 2^{-\rho} \le (1+\Delta)r + 2^{-\rho},$$

which leads to the bounds

$$2(1+\Delta)r \le (1+\Delta)r + 2^{-\rho} \Rightarrow \rho \le -\lg(1+\Delta) - \lg r,$$

and hence

$$\lambda/n - \lg(n)/n - 2 + \tau \le -\lg(1+\Delta) - \lg r \Rightarrow \lambda \le 2n - n\lg(1+\Delta) - n\lg r + \lg n + n\tau.$$

So we must assume that

$$\lambda > 2n - n\lg(1+\Delta) - n\lg r + \lg n + n\tau. \tag{3.5}$$

Both estimates (3.4) and (3.5) should hold in order to imply the desired bounds for r_s.

Next we bound r. Notice that $\frac{r}{2(1+\Delta)} \le r_s \le 2^\tau$ for all s since by assumption all the roots of f lie in the unit disc. Thus $\lg r \le 1 + \lg(1+\Delta) \Rightarrow -\lg r > -1 - \lg(1+\Delta)$.

By combining this inequality and (3.4) deduce that $\lambda > 2n + \lg n$, and so if $\lambda > 2n + \lg n$, then a solution to Task S for $\widetilde{f}$ and $1 + \Delta \ge 2n$ implies a solution to Task S for f, under (3.2).

Next we apply Dandelin's root-squaring iterations in order to decrease the assumed bound on Δ. Every iteration squares the root radii, and so the bound for $1 + \Delta < 2n$ fulfilled after k iterations implies the bound for $1 + \Delta < (2n)^{1/2^k}$ beforehand.

Assume that we have applied k iterations, where k is such that $2^k = N$. Let f_k be the resulting polynomial. Since we work with approximations, it holds that $\|f_k - \widetilde{f}_k\|_\infty \le 2^{-\lambda + O(N\tau + Nn)} = 2^{-\lambda_k}$ according to Lemma 3.2.

At this stage we bound $\widetilde{r}_s^{(k)}$ by computing an r such that $r(1+\Delta) \le \widetilde{r}_s^{(k)} \le r(1+\Delta)$.

If $\lambda_k > 2n + \lg n$, then $r/2(1+\Delta) \le r_s^{(k)} \le 2r(1+\Delta)$, and therefore

$$(r/2(1+\Delta))^{1/2^k} \le r_s \le (2r(1+\Delta))^{1/2^k}.$$

So it suffices to consider a λ such that $\lambda > O(N\tau + Nn) + 2n + \lg n$. A term $c \cdot n$ should be added, for a small constant c, in order to compensate for the computation of the logarithms, the divisions and the n-th roots. This does not alter the asymptotic bound.

As this is estimated in [MP13, Eq. (15.22), (15.23)], we can choose $k = O(\lg n)$ and thus can choose λ of order $O(n\tau + n^2)$.

The number of flops in every Dandelin's iteration loop is $O(n(\lg(n))^2) = \widetilde{O}(n)$ (cf. Theorem 2.2). Thus we can solve Task S in $\widetilde{O}_B(n^3 + n^2\tau)$ for a given s. Recall that the estimate of Theorem 2.4 on the number of flops applies to the solution of Task S for all s, that is, to the approximation of the radii of all the n roots of f. Hence, for this task the bound is $\widetilde{O}_B(n^3 + n^2\tau)$ as well. □

Remark 3.2. *If the input polynomial f is known exactly, for example, if it has rational coefficients, then we can omit ℓ in the bounds of the previous lemma.*

Remark 3.3. *The shift of the variable by $\sigma = 2^l$ implies the growth of the coefficient length τ by $O(nl)$, and we can extend our cost and error bounds for the approximation of the distances of the roots from a point s accordingly.*

3.2 The complexity of the Newton iterations

Having computed sufficiently good approximations $\tilde{r}_j$ to the root radii r_j, we can apply Newton's iterations to all $2n$ candidates $\pm\tilde{r}_j$, $j = 1, \dots, n$ in order to approximate the r real roots to a sufficient accuracy.

If we assume that the intervals containing the real roots have a "proper" isolation ratio, then the cost of the application of Newton's operator is given by the following proposition, which is a slight modification of Lemma 10 and Remark 11 in [PT13, PT14c], see also [PT14b].

Proposition 3.1. *Let $f \in \mathbb{C}[x]$ be of degree n such that $\|f\|_\infty \le 2^\tau$. Assume that we are given a λ-approximation of f, $\widetilde{f}$, such that $\|f - \widetilde{f}\|_\infty \le 2^{-\lambda}$, where $\lambda = \ell + L + O(n\tau)$, $\ell > 0$.*

The maximum number of bits needed by Newton iterations is $\widetilde{O}(L + n\tau + \ell)$, and the total complexity of the Newton step is $\widetilde{O}_B(n^2\tau + nL + n\ell)$.

The same asymptotic bound holds if we apply Newton operator to approximate all the roots simultaneously because each Newton step consists of an evaluation of the polynomial and its derivative. We can perform all these operations simultaneously by using multipoint evaluation [PT14a, Lemma 21] at the same asymptotic Boolean cost [PT14c, Theorem 14].

Newton's iterations converge with quadratic rate right from the start provided that the root and its initial approximation lie in the same $3n$-isolated disc [T98]. How can we test whether this assumption holds for a given polynomial f and a real interval I? We can choose among various known proximity tests (see Section 2.2), and in our case a natural strategy is to just apply the Newton operator. In this way we compute a sequence of real inclusion intervals $(a_h \,..\, b_h)$, for $h = 0, 1, \dots,$ where $(a_0 \,..\, b_0) = I$ and $b_h > a_h$ for all h. We verify the inclusion property by checking whether $f(a_h)f(b_h) < 0$ and either observe that h bisection steps decrease the width of the isolating interval by a factor of 2^h or otherwise conclude that the assumption on the isolation ratio is certainly violated. This *test by action* requires negligible extra cost.

Remark 3.4. *The paper [PT14a] provides a simple recipe for increasing the isolation ratio of a disc from* $1+1/\log_2(n)$ *to* $3n$ *or even to* cn^d *for any pair of real constants* c *and* d *at very low arithmetic and Boolean cost.*

4 Tests for Real Root-finding with Algorithm 2.1

Our tests show that Algorithm 2.1 works quite well on polynomials without clustered roots. At the first stage of this algorithm, we combined the root-radii estimate in [BR14] (based on the algorithms of [B96] and [BF00]) with the numerically stable variant of Dandelin's root-squaring iteration from [MZ01], which exploits tangential representation and renormalization of coefficients.

We run numerical tests on polynomials of two types having degree $n = 64, 128, 256, 512, 1024$, and we compared our results with the outputs of MATLAB function "roots()":

I. $p(x) = p_1(x)p_2(x)$, where $p_1(x)$ is the r-th degree Chebyshev polynomial, $r = 8, 12, 16$, $p_2(x) = x^{n-r} - 1$.

II. $p(x) = p_1(x)p_2(x)$, where $p_1(x)$ is the r-th degree Chebyshev polynomial, $r = 8, 12, 16$, $p_2(x) = 1 + 2x + 3x^2 + \cdots + (n-r+1)x^{n-r}$.

The following tables display the number of iteration and the error bounds when we applied Algorithm 2.1 to polynomials of these two types.

In many cases the number of iterations was small, and then reliable results can be expected even without renormalization. In such cases application of FFT-based polynomial convolution would decrease the quadratic arithmetic complexity of an iteration to $O(n\log(n))$.

If our estimates showed that the norm of the root lied in the range $[\alpha, \beta]$ and if the root is real, then it must lie in one of two intervals: $[-\beta, -\alpha]$ and $[\alpha, \beta]$. For each of them we made a search for a subinterval where the polynomial changed its sign. When we found r such subintervals, we output the number of iterations required for this, then applied five Newton's iterations initiated at the r midpoints, compared their outputs with the roots computed by MATLAB root-finding function "roots()", and output the maximum error bound.

Table 4.1: Number of Iterations and Error Bounds for Algorithm 2.1 on Type I Polynomials

n	**r**	**Iterations**	**Errors**
64	8	9	$5.57E-15$
64	12	10	$1.06E-13$
64	16	9	$5.33E-12$
128	8	9	$5.55E-15$
128	12	11	$6.66E-14$
128	16	11	$1.30E-12$
256	8	10	$1.02E-14$
256	12	11	$3.14E-14$
256	16	11	$2.37E-12$
512	8	10	$3.56E-14$
512	12	11	$4.74E-13$
512	16	12	$3.56E-12$
1024	8	11	$3.01E-14$
1024	12	12	$3.09E-14$
1024	16	13	$1.50E-12$

Table 4.2: Number of Iterations and Error Bounds for Algorithm 2.1 on Type II Polynomials

n	**r**	**Iterations**	**Errors**
64	8	5	$1.01E-14$
64	12	6	$1.32E-13$
64	16	8	$2.26E-12$
128	8	6	$6.32E-15$
128	12	7	$1.31E-13$
128	16	8	$2.20E-12$
256	8	6	$7.10E-15$
256	12	7	$1.93E-14$
256	16	8	$4.49E-12$
512	8	6	$2.33E-15$
512	12	8	$1.95E-14$
512	16	8	$1.35E-12$
1024	8	7	$1.45E-14$
1024	12	8	$2.90E-14$
1024	16	9	$2.19E-12$

References

[BT90] Ben-Or, M., Tiwari, P.: Simple algorithms for approximating all roots of a polynomial with real roots. J. Complexity, 6(4), 417–442 (1990)

[B96] Bini, D.: Numerical computation of polynomial zeros by means of Aberth's method. Numerical Algorithms, 13, 179–200 (1996)

[BF00] Bini, D. A., Fiorentino, G.: Design, analysis, and implementation of a multiprecision polynomial rootfinder. Numerical Algorithms, 23, 127–173 (2000)

[BP94] Bini, D., Pan, V. Y.: Polynomial and Matrix Computations, Volume 1: Fundamental Algorithms. Birkhäuser, Boston (1994)

[BR14] Bini, D.A., Robol, L.: Solving secular and polynomial equations: a multiprecision algorithm. J. Computational and Applied Mathematics, 272, 276–292 (2014)

[C91] Carstensen, C.: Inclusion of the roots of a polynomial based on Gerschgörin theorem. Numerische Math. 59, 349–360 (1991)

[DJLZ97] Du, Q., Jin, M., Li, T.Y., Zeng, Z.: The quasi-Laguerre iteration. Math. Computation, 66(217), 345–361 (1997)

[EPT14] Emiris, I. Z., Pan, V. Y., Tsigaridas, E.: Algebraic algorithms. Chapter 10 of Computing Handbook (Third edition), Volume I: Computer Science and Software Engineering (Allen B. Tucker, Teo Gonzales, and Jorge L. Diaz-Herrera, editors). Taylor and Francis Group (2014). Available at arXiv 1311.3731 [cs.DS]

[F02] Fortune, S.: An Iterated Eigenvalue Algorithm for Approximating Roots of Univariate Polynomials. J. of Symbolic Computation, 33(5), 627–646 (2002)

[GL13] Golub, G.H., Van Loan, C.F.: Matrix Computations, 4th edition. The Johns Hopkins University Press, Baltimore, Maryland (2013)

[H59] Householder, A.S.: Dandelin, Lobachevskii, or Graeffe. Amer. Math. Monthly, 66, 464–466 (1959)

[MZ01] Malajovich, G.,. Zubelli, J. P.: Tangent Graeffe iteration. Numerische Mathematik, 89 (4), 749–782 (2001)

[M07] McNamee, J.M.: Numerical Methods for Roots of Polynomials. Part 1 (XIX + 354 pages), Elsevier (2007)

[MP13] McNamee, J.M., Pan, V.Y.: Numerical Methods for Roots of Polynomials. Part 2 (XXII + 718 pages), Elsevier (2013)

[P95] Pan, V.Y.: Optimal (up to polylog factors) sequential and parallel algorithms for approximating complex polynomial zeros. In: Proc. 27th Ann. ACM Symp. on Theory of Computing, pp. 741–750. ACM Press, New York (1995)

[P00] Pan, V.Y.: Approximating complex polynomial zeros: modified quadtree (Weyl's) construction and improved Newton's iteration. J. of Complexity, 16(1), 213-264 (2000)

[P01] Pan, V.Y.: Structured Matrices and Polynomials: Unified Superfast Algorithms. Birkhäuser, Boston, and Springer, New York (2001)

[P01a] Pan, V.Y.: A new proximity test for polynomial zeros. Computers and Math. (with Applications), 41(12), 1559-1560 (2001)

[P02] Pan, V.Y.: Univariate polynomials: nearly optimal algorithms for factorization and rootfinding. J. Symb. Computations 33(5), 701–733 (2002). Proc. version in ISSAC'2001, pp. 253–267, ACM Press, New York (2001)

[PT13] Pan, V.Y., Tsigaridas, E.P.: On the Boolean complexity of the real root refinement. Proc. Intern. Symposium on Symbolic and Algebraic Computation (ISSAC 2013), (M. Kauers editor), pp. 299–306, Boston, MA, June 2013. ACM Press, New York (2013) Also arXiv 1404.4775 April 18, 2014

[PT14a] Pan, V.Y., Tsigaridas, E.P.: Nearly optimal computations with structured matrices. In: Proc. of the International Conference on Symbolic Numeric Computation (SNC'2014). ACM Press, New York (2014). Also April 18, 2014, arXiv:1404.4768 [math.NA]

[PT14b] Pan, V.Y., Tsigaridas, E.P.: Accelerated approximation of the complex roots of a univariate polynomial. In: Proc. of the International Conference on Symbolic Numeric Computation (SNC'2014). ACM Press, New York (2014). Also April 18, 2014, arXiv : 1404.4775 [math.NA]

[PT14c] V. Y. Pan and E. Tsigaridas. Nearly Optimal Refinement of Real Roots of a Univariate Polynomial. Report 2014.

[R87] Renegar, J.: On the worst-case arithmetic complexity of approximating zeros of polynomials, J. of Complexity 3(2), 90–113 (1987).

[S82] Schönhage, A.: The fundamental theorem of algebra in terms of computational complexity. Math. Department, Univ. Tübingen, Germany (1982)

[SMa] Sagraloff, M., Mehlhorn, K.: Computing real roots of real polynomials. CoRR, abstract 1308.4088 (2013)

[T98] Tilli, P.: Convergence conditions of some methods for the simultaneous computations of polynomial zeros. Calcolo, 35, 3–15 (1998)

[VdS70] Van der Sluis, A.: Upper bounds on the roots of polynomials. Numerische Math. 15, 250–262 (1970)